

Architecture Definition

Riverton defines software architecture to be the structure (or structures) of a system (both business and technical), which is comprised of software services, the externally visible properties of these services, and the relationship among them. Any architecture is the result of technical, business, and social influences. Technical influences drive the creation and realization of the architecture. Business influences drive the requirements and definition of the problem set. Social influences determine the architecture from the basis of the environment and the people involved (end users, development team, sponsors, etc.).

Software Architecture:

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

These three influences all contribute to the definition of software architecture. A good architecture satisfies the behavioral, technical, business, performance, and life cycle requirements that are defined before the creation of the architecture. An overall good architecture is founded upon the interactions between a business architecture and an [Enterprise Architecture](#).

In addition to responding to these influences, a “good” architecture must meet certain goals and principles that are generally not reflected in these influences. This white paper describes our perspective on these goals and principles.

Architecture Principles

Given this definition, there are a number of factors to consider while designing the architecture. Our underlying goals for a good architecture include:

- Scalable* Scalability implies the ability for the architecture to grow and accommodate increasing numbers of users, applications, and systems. The architecture must also accommodate changing business processes, migrating data, and new uses of existing data.
- Flexible* Flexibility implies the ability for the architecture to remain vendor and technology independent. Technology will always influence the structure of the architecture but in no means should constrain the architecture.
- Extensible* The architecture should be able to extend from its first implementation. It should grow and accommodate new business requirements as they are determined and incorporated into the architecture.

- Reusable* The architecture should foster reuse wherever possible. Equally important is the underlying effect that implementation of the architecture encourages the development of reusable components.
- Open* The architecture should incorporate open standards wherever possible. Open standards contribute to implementing many of the other principles of a good architecture
- Secure* Today's environment demands that an architecture appreciates and handles all manners of security on data and business processes.
- Semantics* Common semantics is a key goal of an architecture since it enhances the ability of an architecture to maintain many of the other goals. Common semantics allows an architecture to provide abstraction (at the business level) and communicate with different systems and business processes in a maintainable manner (at the technical level).

Additionally, a good architecture is designed with the following principles:

- At the application level, the architecture is comprised of well-defined services with functional interfaces that enforce the principles of information hiding, encapsulation, and separation of concerns.
- Separation of concerns is a focus of the architecture because it allows development teams to work independently of each other.
- Tools and approaches should be chosen for suitability to purpose, and incorporate best of breed technologies wherever appropriate.
- Encapsulation and information hiding allow the services to insulate applications from platform dependencies.
- The architecture should not depend on a particular version of a product or tool.
- Design services that promote the separation of data, business logic, and presentation so that tight coupling among the three is avoidable.
- Patterns should be applied where appropriate. Patterns offer successful approaches to solving common architectural issues with a structured, well-defined solution.

Shared services or in modern parlance, web services, provide reusable building blocks of components that can be extended through out an enterprise. They are self-contained, self-describing, modular components that can be published, located, and invoked across the network.

A pattern describes best practices, good design, and captures experience in a way that is possible for others to reuse.

Reusability

A good architecture is designed to allow the utmost in reusability and separation of talents required to work on a particular subcomponent of the larger design. The most prominent example is the separation of data access from the business logic and from the display logic. This allows re-use at all those levels. Each of those layers can be re-used in multiple applications.

An architecture should also put heavy emphasis on shared services. These resources, used at run time, are by definition re-usable by multiple running applications and services. Shared services can include a business service, security service, persistence service and many more. Any piece of functionality that can be re-used at run time is amenable to being vended as a service.

Reuse can also be approached at the design level through the use of patterns. Patterns are a mechanism for capturing existing, well-proven expertise in software development. The use of patterns promotes consistency and quality through the course of design and implementation. Effective combinations of patterns that solve specific problems allow scalable and flexible architectures to be constructed.

Common, fundamental attributes of patterns include:

- A pattern addresses a recurring design problem that arises in specific situations, and presents a solution to it;
- Patterns document existing, well proven design experience
- Patterns identify and specify abstractions that are above the level of single classes and instances or components
- Patterns provide a common vocabulary and understanding for design principles
- Patterns are a means of documenting software architectures
- Patterns support the construction of software with defined properties
- Patterns contribute to building complex and heterogeneous software systems
- Patterns help manage software complexity

There are two main classifications for software patterns, Architectural Patterns and Design Patterns. An architectural pattern describes a structural organization for a system with the rules and guidelines for managing the relationships between pre-determined components. Design patterns on the other hand focus more on the definition of the components that contribute to an architectural pattern. A good architecture incorporates both types of patterns to illustrate concepts where appropriate.

Scalability

Scalability is defined as the ability of a technique or algorithm to work when applied to larger problems or data sets (*Academic Press Dictionary of Science and Technology*). It also implies the notion of how well a solution to some problem will work when the size of the problem increases. In today's distributed environments, one approach to scalability translates into the ability to distribute functionality across multiple hardware components or software components. There are many other approaches to address scalability. A good architecture supports functionality (and hence scalability) being broken off into services. The value in the service-based approach is that one can easily migrate to larger hardware or duplicate the functionality of a service without necessarily affecting the rest of the system. In addition, the layering is such that tools like TP monitors and object request brokers (ORB's) can be introduced without breaking the architecture. A Data Abstraction Layer (DAL) allows new database management systems to be transparently introduced that may have different performance and scalability profiles.

A Data Abstraction Layer (DAL) provides a mechanism for decoupling the physicality of data from its use in the business logic of an application.

Extensibility and Flexibility

The key to extensibility in architecture is modularity. By providing true abstractions and clear boundaries between layers, new technologies and tools that provide better functionality can be substituted for aging, less effective ones. Other essential concepts include a decoupled approach to architecture, the incorporation of services, use of technologies that support these concepts and a foundation for addressing integration (as opposed to point to point solutions). Sophisticated languages such as Java have run-time support for adding functionality on the fly. Very often, the application can even be running when the new functionality is added.

Common Semantics

Common semantics get the business organizations and the technical organizations talking the same language. At the design stage, the Unified Modeling Language (UML) provides a structured set of documents that facilitates communication among the business and technology organizations. During an implementation effort, the Extensible Markup Language (XML) provides a way of structuring data that allows independence from the data's locality and physical definition.

XML is a markup language for documents containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays. For example, content in a section heading has a different meaning from content in a footnote, which means something different from content in a figure caption etc. Almost all documents have some structure.

XML is a meta programming language developed by the W3C to allow developers to create customized tags to organize and deliver content more efficiently.

A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents. For our purposes, the word "document" refers not only to traditional documents, like this one, but also to the assortment of other potential XML "data formats". These include business objects, vector graphics, e-commerce transactions, mathematical equations, object meta-data, server APIs, etc. In HTML, both the tag semantics and the tag set are fixed. An <h1> is always a first level heading. The W3C, in conjunction with browser vendors and the WWW community, is constantly working to extend the definition of HTML to allow new tags to keep pace with changing technology and to bring variations in presentation (i.e. style sheets) to the Web. However, these changes are always rigidly confined by what the browser vendors have implemented and by the fact that backward compatibility is paramount. For people who want to disseminate information widely, XML is a more logical choice since it is self describing and extensible.

XML specifies neither semantics nor a tag set. XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there's no predefined tag set, there can't be any preconceived semantics. All of the semantics of an XML document will be defined either by the applications that process them or by XML Schemas and Document Template Definitions (DTD's).

A good architecture employs XML in a number of areas. It becomes a common semantic for communication among various layers of the architecture. For example, the messaging layer should be built completely in XML to allow easy transition from all back end systems to the front end. Supplying a common model and representation of the data in the backends supplies the architecture the flexibility to grow, as business needs change. Another area of the architecture that should make heavy use of XML is in the client communication layer. Although a particular protocol (HTTP, HTTPS, IIOP, RMI, etc.) is used for the distribution, the encoding should be in XML.

Summary

A good architecture incorporates the principles and goals we have outlined in this paper. Many external factors influence the adoption and implementation of a good architecture and it is up to an organization to undergo the cultural and technical migrations in order to support the architecture in an ongoing fashion. An architecture founded on these principles will grow and mature with an organization as it grows and matures.

About Riverton Corporation

Riverton is a specialized, consulting and integration company focused on helping companies solve business challenges through the effective application of advanced technology since 1995. It provides enterprise architecture, cross-enterprise integration, custom application development, and business intelligence solutions. It has strong expertise with open standards such as J2EE and XML and incorporates industry best practices like RUP and UML in its delivery lifecycle. For more information on Riverton service offerings go to www.riverton.com or send e-mail to sales@riverton.com.